"Express Mail" mailing label number EL485651283US
Date of Deposit December 19, 2000

Atty Docket No. 00P7662US01

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

This is a U.S. Patent Application for:

TITLE: SYSTEM AND METHOD FOR ANALYZING AND GENERATING SUPPLEMENTARY SERVICE DATA UNITS IN PACKET BASED MULTIMEDIA COMMUNICATIONS SYSTEMS

Inventor #1: Florian Trinkwalder
Address: 534 Giuffrida Avenue, #A, San Jose, California 95123
Citizenship: Germany

Inventor #2: Mark E. Clark
Address: 2986 Little Rock Drive, San Jose, California 95133
Citizenship: USA

Inventor #3: Mark Skrzynski
Address: 1800 42nd Avenue, Capitola, California 95010
Citizenship: USA

SYSTEM AND METHOD FOR ANALYZING AND GENERATING SUPPLEMENTARY SERVICE DATA UNITS IN PACKET BASED MULTIMEDIA COMMUNICATIONS SYSTEMS

CROSS REFERENCE TO RELATED APPLICATIONS

This application is a continuation-in-part of commonly owned U.S. Patent Application Serial No. 09/586,408, filed June 1, 2000, and titled "System and Method for Sending Messages Defined by an ASN.1 Structure from an Application to the Signaling Entity and Vice Versa Via a Tree Structured Text String," which is hereby incorporated by reference in its entirety as if fully set forth herein.

RESERVATION OF COPYRIGHT

BACKGROUND OF THE INVENTION

FIELD OF THE INVENTION

The present invention relates to telecommunication systems and, particularly, to a system and method for sending messages from an application to signaling entities and, for particular embodiments, the sending of APDUs between H.323/H.450 endpoints.

DESCRIPTION OF THE RELATED ART

The Abstract Syntax Notation One (ASN.1), described in the International Telecommunications Union (ITU) X.680 and X.691 specifications, defines a data structure protocol for describing messages to be exchanged between distributed computer systems. ASN.1 defines data units independently of system architecture. Because of this, ASN.1 is used to generate programming language code that forms the core of a wide variety of

messaging systems applications, including ITU-T Recommendation H.323.

The ITU-T Recommendation H.323 is a group of specifications defining the operation of a multimedia communication system over packet networks. The ITU-T Recommendation H.323 specifies H.245 control signaling for negotiation of media channel usage, Q.931 (H.225.0) for call signaling and call setup, H.225.0 Registration, Admission, and Status (RAS), H.450 for supplementary services, and RTP/RTCP for sequencing audio and video packets. An exemplary system implementing the Recommendation H.323 is the HiPath™ 5500 system, available from Siemens Information and Communication Networks, Inc.

H.323-based systems use ASN.1 coded messages and protocol state machines for describing the application protocol data units (APDU) or packets of data used for signaling between H.323 endpoints, servers, gateways, and gatekeepers.

The processing of H.323 ASN.1 messages is typically implemented in a protocol stack, and its functions are accessed by application programs through application programming interfaces (API). The software primitives of the APIs are typically written with programming languages, such as C, C++, and Java, and all the associated parameters are expressed and declared in the corresponding language.

Whenever new features, such as new parameters or completely new supplementary services are added to the protocol stack of a software product, such as an H.323 application, software changes must be done in three layers: new functional entities must be added to the protocol stack; new parameters or programming primitives must be added to the API; and applications and their user interfaces must be changed or enhanced to allow user access to the new features. Each operation carried out by the new feature must be callable via an API that must be explicitly exported to the application. The changes must be made at the source code level, and the new software system must be recompiled and linked together and loaded to the customer's target system.

Thus, changes and additions to the software require replacement of

the software in the target system. While software can be downloaded remotely, an interruption of the target system's operation occurs, which can hinder the simple "plugging in" of new features to the system.

Moreover, unless the API is modified, the application cannot access new features. Thus, even relatively small changes and additions to the signaling protocols and related protocol stacks require relatively high development effort, since the API source code must be modified.

Therefore, there is a need for an improved method for updating the protocol stack of an API based software product. There is particularly a need for an improved method for updating the processing of ASN.1 messages in an H.323 telecommunications system.

Further, as is known, H.225 messages are used for call signaling to establish a connection between endpoints. Initially, an H.225.0 RAS (registration, admission, status) channel is established between an endpoint and a gatekeeper (in the gatekeeper-routed call model). After the RAS signaling channel is established, an H.225.0 call signaling channel is established between endpoints. H.225.0 call signaling messages include SETUP, ALERTING, CONNECT, RELEASE COMPLETE, and FACILITY. The user-user information element of an H.225 message can carry an H.450 APDU (application protocol data unit) for H.450 supplementary services. Such services are call-related services beyond the basic call and can include Call Forwarding, Call Transfer, Call Waiting, Message Waiting Indication, Conference, and the like.

As can be appreciated, newly implemented H.450 supplementary service features must be tested with all possible normal and exceptional events. To perform these tests, various H.450 messages must be sent and received by the endpoints. As such, there is a need for a system and method for sending, receiving, and displaying such test messages.

SUMMARY OF THE INVENTION

These and other problems in the prior art are overcome in large part by a system and method according to the present invention. According to the

present invention, signaling messages and parameters that are defined by an ASN.1 structure are sent as a tree-structured text string from an application to a signaling entity (SE) and vice versa. As new signaling entities are added to support new features, the features can receive and send primitives without the programmer having to change or expand the API.

According to an implementation of the present invention, application programming interfaces (APIs) are provided which implement a first recursive function for interpreting text strings representative of ASN.1 structures for communicating with an application program. A second recursive function receives and interprets ASN.1 value trees from signaling entities (SE).

According to an implementation of the invention, an H.450 test client is provided. The H.450 test client can send and receive H.450 application protocol data units in any H.225 message. The test client allows for the setting of desired H.450 parameters and allows for the display of the H.450 message itself.

BRIEF DESCRIPTION OF THE DRAWINGS

A better understanding of the invention is obtained when the following detailed description is considered in conjunction with the following drawings in which:

FIG. 1 is a block diagram illustrating an implementation of the present invention;

FIG. 2 is a functional diagram illustrating an implementation of the present invention;

FIG. 3 is a flowchart illustrating an implementation of the present invention;

FIG. 4 is a functional diagram of an implementation of the present invention;

FIG. 5 is a flowchart illustrating operation of an implementation of the invention;

FIG. 6 is a graphical user interface according to an implementation of the present invention;

FIG. 7 is a diagram of another aspect of the graphical user interface of FIG. 6;

FIG. 8 is an exemplary H.225 packet;

FIG. 9 and FIG. 10 are exemplary graphical user interfaces according to an implementation of the invention;

FIG. 11 is a flowchart illustrating operation of an implementation of the invention;

FIG. 12A and FIG. 12B are exemplary graphical user interfaces according to an implementation of the invention;

FIG. 13 is an exemplary graphical user interface according to an implementation of the invention; and

FIG. 14 is a diagram of a graphical user interface according to an implementation of the invention.

DETAILED DESCRIPTION OF THE INVENTION

FIGs. 1- 14 illustrate an improved system and method according to the present invention. A first implementation is a system and method for configuring a communication system with system updates without having to recompile associated application programming interfaces. In a second implementation, an H.450 test client is provided. The H.450 test client can send and receive H.450 application protocol data units in any H.225 message. The test client allows for the setting of desired H.450 parameters and allows for the display of the H.450 message itself.

ASN.1 Text Strings

An exemplary computer system implementing a communication system according to an implementation of the invention is shown in FIG. 1. As will be described in greater detail below, the system 102 may be embodied as an H.323 terminal. In particular, the system may be embodied as a personal computer, such as an x86 compatible personal computer or an Apple Macintosh G4. The computer 102 includes a processor 11 adapted to implement computer code according to the present invention. Further,

according to an implementation of the present invention, the processor 11 implements an operating system (not shown) for generating a graphical user interface (GUI) 520 to control system operation.

As shown in FIG. 1, an application program 502, such as a telephony application, one or more APIs 506, and one or more signaling entities 510a-510n are resident. As will be explained in greater detail below, the APIs 506 implement recursion functions 508 that allow ASN.1 text strings to be passed to the application 502 and the signaling entities 510. Thus, as will be explained in greater detail below, when upgrading the system, i.e., when new signaling entities are provided, the APIs need not be altered.

More particularly, FIG. 2 is a functional diagram of the system according to the present invention. The system includes an application program 502, such as a telephony program. The application program 502 communicates via text strings 503a, b with one or more APIs 506, as will be explained in greater detail below. The text strings 503a,b contain the ASN.1 tree structure of the message and the corresponding values.

The ASN.1 tree structure is defined in terms of "valuetrees" and "syntaxtrees." The syntaxtree is static and contains the syntax of all possible information elements of a message structure. The valuetree is dynamic and contains all information elements that exist for a specific message. The value tree also contains the values of the various fields.

In one implementation, the structure of the message, i.e., the syntaxtree is provided to the application program 502 ahead of time. In that case, although the application 502 and the corresponding signaling entity 510 must be updated in order to add new features, the API 506 remains the same.

In an alternate implementation, the structure and syntax of the message are provided to the application program 502 from the stack (not shown). In that case, the application program 502 will first call the corresponding API to retrieve the syntaxtree structure for the message from the stack. In this case, to add new features, both the application program 502 and the API 506 could be left unchanged. The only change then required to

the system is new signaling entity script.

In operation, a first recursive function 508a receives the syntaxtree text strings 503a, fills in the valuetrees, and passes them to one or more signaling entities 510. The signaling entities may be any entities adapted to generate encodeable data according to specified functions. In particular, the signaling entities 510 may be embodied as H.450 signaling entities. Such signaling entities include the Message Waiting Indication (MWI), Call Transfer, Call Diversion, Call Hold, Call Waiting Call Completion, Call Pickup, and Number Identification signaling entities. The signaling entities 510 may be implemented in protocol description language (PDL) scripts. Further details on PDL scripts are available from Radvision, Ltd. The PDL scripts contain the state machine code of the signaling entities and also implement machines arranged in a tree structure.

In turn, the signaling entities 510 communicate their respective APDUs (application protocol data unit) for ASN.1 encoding/decoding 512. The ASN.1 encoded octet strings are then provided onto the network 514.

When messages are received from the network, the ASN.1 octet strings are decoded, and the corresponding APDUs are provided to the appropriate signaling entities 510. The APIs 506 then receive the message values from the signaling entities 510, interpret them using a second recursion function 508b, and pass them as text strings to the application.

Operation of an embodiment of the invention is illustrated with reference to the flowchart of FIG. 3. In a step 452, an application program 502 builds up a tree-structured text string, i.e., the valuetree, for a message, such as an H.450 message. In a step 454, the API 506 receives the valuetree string and fills the information elements into the valuetree. In a step 456, the appropriate signaling entity 510 is created according to the operation in the message. In a step 458, the signaling entity sends the message in an H.450 APDU over the network. In a step 460, the H.450 APDU is received from the network by the receiving party at its corresponding remote signaling entity. In a step 462, the remote signaling entity sends the message's valuetree to its API. In a step 464, the API reads out the valuetree and sends

the information elements to the application program in a tree-structured text string. Finally, in a step 466, the application program receives the tree-structured text string and processes the information elements.

Operation of a particular implementation of the present invention may be illustrated by way of a specific example. According to a specific embodiment, code implementing the example follows as an Appendix. Shown in FIG. 4 is a software model of the example. The model includes an MWI Test Utility 502a, i.e., Application Program. The Message Waiting Test utility 502a tests the H.450 message waiting indication supplementary service. The Message Waiting supplementary service indicates on the served user that messages are waiting on the message center.

The model also includes an H.450 Test API 506a, a Message Center Signaling Entity 510a, a Served User Signaling Entity 510c, and a Message Linker 510b. To signal the relevant information over the network, APDUs with activation, deactivation, and interrogation operations are sent to and received from the network by the Served User Signaling Entity 510c and the Message Center Signaling Entity 510a.

The Message Linker 510b functions to provide a distribution process by determining to which signaling entity a specific text primitive or APDU is to be sent. That is, messages are sent to and received from the signaling entities 510a via the Message Linker 510b.

As seen in the Appendix, the H450 Test API 506a employs the files H450api.c, H450api.h, and H450struct.h. The Supplementary Service employs the files MWI.pdl and MWI-operations.asn. The H450 Test API implements initAPI, setCallback, initH450Root, and endH450Root initialization and termination functions. The initAPI function initializes the API. The setCallback function sets the callback. The initH450Root creates and initializes, and the end450Root function terminates, the H450 root protocol.

As is known, when a PDL state machine is executed, the resulting data structure is called a "process." As the PDL engine executes the state machines, the processes are also arranged in a tree structure and the process tree is built in a hierarchical manner. When a process activates

another process, the new process is located under the existing process in the process tree, the new process being defined as a "child" process of the existing process.

In addition to the process tree, the PDL engine builds another tree called the "protocol" tree. A protocol is a data structure that allows interaction between the application and the PDL machines. The protocol tree has a similar structure to that of the process tree, but not all of the elements of the process tree are recreated in the protocol tree.

The Test H450 API 506a (FIG. 4) implements createH450protocol, closeH450protocol, newH450RootChild, and newH450Child protocol functions. The createH450protocol function creates a new H450 protocol under the protocol root. The closeH450protocol closes the H450 protocol. The newH450RootChild is called when a new protocol is created under the protocol root. The newH450Child function is called when a new child is created under the message linker protocol.

The H450 test API 506a implements the following message functions: sendmessage450, newH450message, fillin, and recursion. The sendmessageH450 function is called when a new message is sent from the application to the H450 protocol. The newH450message is called if the applsend function is executed in the PDL script in order to send a message back to the application.

The fillin() and recursion() function to transform the tree-structured text-strings into valuetree structures and vice versa. The fillin() function receives the string as a parameter and fills in the various fields in the valuetree (XMLstring -> valuetree). The recursion() function reads out the valuetree and parses its nodes and values into a textstring (valuetree -> XMLstring).

FIG. 5 is a flowchart illustrating operation of the embodiment of the invention. In a step 650, the application program builds up a tree-structured text string for an H.450 message. In a step 652, the API receives the text string and fills in the information elements into the value tree using the fillin() function and creates the Message Linker protocol, i.e., the root protocol. In a

step 654, the Message Linker creates the Signaling Entity according to the operation in the message. In a step 656, the Signaling Entity sends the message in an H.450 APDU over the network. In a step 658, the Message Linker receives the H.450 APDU from the network and creates the Signaling Entity according to the operation received. In a step 660, the Signaling Entity receives the APDU from the Message Center and sends its value tree to the API. In a step 662, the API reads the valuetree using the recursion() function and sends the information elements to the application in a tree-structured text string. Finally, in a step 664, the application receives the tree-structured text string and processes the information elements.

As noted above, one aspect of implementing the invention is a graphical user interface. FIG. 6 illustrates an exemplary graphical user interface 520 for the message waiting indication (MWI) test utility. As shown, the graphical user interface 520 includes a message center (MC) test control 524 and a served user test control 526. The GUI 520 further includes a text display region 522, used to display the ASN.1 text strings. The message center test control 524 includes an Activate Request button 528, a Deactivate Request button 530, an Interrogation Response (ack) button 532, and an Interrogation Response (rej) button 534. The served user test control 526 includes an Activate Response (ack) button 536, an Activate Response (rej) button 538, a Deactivate Response (ack) button 540, a Deactivate Response (rej) button 542, and an Interrogation Request button 544. The Message Waiting Indication (MWI) service activates an indication device (e.g. LED) on the served user endpoint if a message (e.g.. voicemail) is waiting on the message center. Three operations can be sent over the network: Activation (to activate the MWI (turn on LED)); Deactivation (to deactivate the MWI (turn off LED)); Interrogation (the served user queries its MWI status).

The "XXX Request Buttons" (where XXX stands, e.g., for Activation) request to send the corresponding operation in an H.450 APDU to the peer entity. If the peer entity receives an indication of an operation (e.g. activation), it will respond by either acknowledging the request by pressing

the "XXX Response (ack) Button" or by rejecting the request by pressing the "XXX Response (rej) Button".

In operation, a user can push a button, such as Activate Request 528. In response, a dialog window 600 (FIG. 7) is displayed. As shown, the dialog window 600 includes a plurality of data fields, and a SAVE button and a SEND button. Shown are servedUserNr, basicService, msgCenterId, nb0fMessages, originatingNr, timestamp, and priority fields. Once filled in, by pressing the SAVE button, the values are stored in a h4507.ini file; by pressing SEND, the message is sent as a text string to the signaling entity.

H.450 Test Utility

As noted above, one aspect of the invention is an ability to send and receive any H.450 protocol data unit (PDU) in any H.225 message. More particularly, FIG. 8 illustrates an exemplary H.225 signaling message 800. As shown, the signaling message 800 includes a TCP header 802 and an H.225.0 V2 message 804. The H.225.0 V2 message 804 is representative of, for example, a SETUP, CONNECT, FACILITY, etc., message. The H.225.0 V2 message 804 includes H.225.0V2/Q.931 information elements 806, setup information element 808 and an H.450 PDU 810. The H.450 PDU 810 includes a network facility extension (NFE) 812, an interpreter APDU (IAPDU) 814, an H.450 feature 816, an MSI extension 818, and an MSI specific feature 820. The NFE 812 defines the type of source and destination of the operation. The IAPDU 814 defines what the receiver is to do if it does not understand the command. The H.450 feature 816 is the APDU itself, and contains an MSI extension which identifies the manufacturer, as does the MSI specific feature 820.

As will be explained in greater detail below, the present invention allows the user to send test H.225 messages, with or without H.450 APDUs. If an H.450 APDU is to be sent, the user can select the specific H.450 APDU and any combination of information elements and network facility extension. The messages are then sent and received, as will be discussed in greater detail below.

A graphical user interface is used to build the H.450 APDU. A graphical user interface in accordance with an implementation of the present invention is shown in FIG. 9. The GUI 900 allows the user to select the H.225 message which is to be tested. For example, the user can select SETUP 904a, ALERTING 904b, or FACILITY 904c buttons to select the corresponding H.225 message. One or more informational fields 902, such as called party and calling party phone number and alias may also be provided. Further, a list field 906 is provided to display the received H.450 message in an XML-like syntax, as generally described above. The list field 906 displays the H.225 message in which the H.450 APDU was received, and also displays all the H.450.1 information elements.

If the user decides to send an H.225 message, e.g., by clicking on SETUP, and after inputting the target number or name, the dialog box 950 of FIG. 10 is displayed. The dialog box 950 allows setting the H.450 information elements. Thus, the dialog box 950 includes an interpretation APDU select dialog 952 and a network facility extension select dialog 954. Using the pulldown 953, the user can select any of a plurality of interpretation APDUs. Alternatively, by deactivating the checkbox 951 the user can elect to send an H.225 message without the IAPDU. Similarly, the user can elect to send no network facility extension, by clicking on the checkbox 955.

The user can input a source and destination entity using the source entity dialog box 956 and destination entity dialog box 958, respectively. The source entity dialog allows the user to enter an entity type 960a and a source entity address 962. The source entity address 962 allows a user to input a value and address type 966a, 968a. As shown, a Party Number type has been selected. The party number dialog 970a allows the user to select a type of party number 972a and then, more specifically, the type of private 974a or public 976a party number. Alternatively, a transport address 978a may be supplied.

Similarly, the destination entity dialog can be used to select an entity type 960b, and includes a destination entity address dialog 964. The destination entity address dialog can be used to input a value 966b and select

a destination entity address type 968b. As shown, a transported type has been selected. The transport ID, e.g., IP address and port, may be input using the dialog 978b. If a party number type had been selected, the user could input a party number type 972b and a private party number type 974b or public party number type 976b.

The entries may be saved by clicking on SAVE 980. Alternatively, the H.225 message may be sent without the H.450 APDU by clicking on button 982 or with the APDU by clicking on button 984.

FIG. 12A and FIG. 12B illustrate graphical user interfaces according to another implementation of the invention. FIG. 12A illustrates a Select H450 APDU Dialog 1200. The Dialog 1200 appears, for example, when Setup (FIG. 9) is clicked. The Select H450 APDU Dialog includes OK button 1202, Cancel button 1204, and New button 1206, as well as List dialog 1208. By clicking on the Cancel button 1204, the user can decide not to send an APDU with this H.225 message, in which case, the Dialog 1200 will close and the message will be sent immediately. Clicking the New button 1206 will start an H450 Builder Application, as will be explained in greater detail below, which allows the building of a new H450 APDU and adding it to the List dialog 1208. After the APDU is created and saved and the user closes the APDU Builder application, the List dialog 1208 is refreshed to show the new APDU.

If the user highlights an APDU in the List dialog 1208 and clicks the OK button 1202, the Select Operation Argument Dialog 1210 (FIG. 12B) appears. The Select Operation Argument Dialog 1210 includes an OK button 1212, a Cancel button 1214, a New button 1216, and a List Dialog 1218.

By clicking the Cancel button 1214, the user can decide not to send an Argument with this APDU, which will close the dialog and send the message immediately. Clicking on New 1216 will start the H450 Builder Application which allows the user to build a new argument and add it to the List dialog 1218. Once the new Argument is created and the Builder is exited, the List 1218 is refreshed.

If the user highlights an Argument in the List 1218 and clicks the OK button 1212, the H.450 Client processes all necessary RAS procedures and

send the setup message together with the H.450 APDU to the specified destination.

Operation of the H450 Builder is shown in greater detail with reference to FIG. 13 and FIG. 14. More particularly, FIG. 13 illustrates a Create APDU dialog 1300. The Create APDU dialog 1300 includes an OK button 1302, a Cancel button 1304 and an entry dialog 1306. If the user presses Cancel 1304, the application terminates.

Otherwise, at startup, the Builder first asks what type of APDU the user wants to create. The user does so by typing in the appropriate APDU in the Entry dialog 1306. The user can type in "h4501" to build an H.450 APDU or "h450x" to build an argument, where x refers to the number of the H.450 recommendation. When the user clicks "OK" 1302, the Builder GUI (FIG. 14) appears.

The APDU Builder 1400 displays a tree control, which displays the whole syntaxtree of the H450 APDU. By clicking on the nodes of the tree, the user can select or deselect nodes for building. If a leaf node (i.e., an end node) is selected, a dialog box (not shown) appears for the user to fill in data.

For example, FIG. 14 illustrates a syntaxtree for an H4501 supplementary service. The user can select network facility extension, InterpretationAPDU, and serviceAPDU nodes. If the user selects networkfacilityextension, the user can choose a SourceEntity, a SourceEntityAddress, and a DestinationEntity and DestinationEntityAddress. If the user selects destinationEntityAddress, a variety of choices are available, such as H323-ID, url-ID, and the like. The user can select one and fill in the appropriate value.

Turning now to FIG. 11, a flowchart illustrating operation of an implementation of the invention is shown. In step 1000, the user uses GUI 900 (FIG. 9) to select a particular H.225 message for sending. In a step 1002, a selection of the IAPDU may be made, using the checkbox 951 and the dialog 953 (FIG. 10) or the Select APDU, Select Operation Argument dialogs (FIG. 12A-12B). In a step 1004, a selection of the Network Facility Extension may be made using the checkbox 935 and the dialog 954 or the

Builder 1400. As discussed above, this can include selection of a variety of source and destination entities, and addresses, including party numbers or transport IDs. In a step 1006, the user can elect to save the filled in information or send the H.225 message with or without the APDU, using buttons 980, 984, or 984 (or the dialogs 1200, 1210, 1300, 1400). In a step 1005, the H.450 IAPDU is displayed.

The invention described in the above detailed description is not intended to be limited to the specific form set forth herein, but is intended to cover such alternatives, modifications and equivalents as can reasonably be included within the spirit and scope of the appended claims.

APPENDIX

```
#
#
# Value encodings:
#    ''    -  String (and asciiz is not appended)
#    ""    -  BMP string of ASCII charactes
#    []    -  Hex octet string
#    <>    -  IP
#    {}    -  Object ID
#   Other -  Integer


1 system = 0
2  allocations = 0
3   vtPoolSize = 100000
3   vtNodeCount = 30000
3   channels = 800
3   chanDescs = 5
3   messages = 400
3   nameChans = 10
3   tpktChans = 25
3   udpChans = 5
3   protocols = 800
3   maxProcs = 800

1 RAS = 0
2  responseTimeOut = 20
2  allowCallsWhenNonReg = 0
2  manualRegistration = 0
2  manualDiscovery = 0
3   defaultGatekeeper = 0
4    ipAddress = 0
5     ip = [c8c8c8c8]
5     port = 1719

2  registrationInfo = 0
3   terminalType = 0
4    vendor = 0
5     vendor = 0
6      t35CountryCode = 11
6      t35Extension = 11
6      manufacturerCode = 11
5     productId = 'Test application'
5     versionId = 'RADVision'
4    terminal = 0
4    mc = 0
4    undefinedNode = 0
3   terminalAlias = 0
4    * = 0
5     e164 = '12345'

2  rasMulticastAddress = 0
3   ipAddress = 0
4    ip = <224.0.1.41>
4    port = 1718
2  rasPort = 0

1 Q931 = 0
2  manualAccept=0
2  responseTimeOut = 150
2  connectTimeOut = 500
2  callSignalingPort = 1720
2  maxCalls = 10

1 h245 = 0
2  masterSlave = 0
```

```
3   terminalType = 50
3   timeout = 100
2  capabilities = 0
3   timeout = 100
3   terminalCapabilitySet = 0
4    sequenceNumber = 0
4    protocolIdentifier = [00]
4    multiplexCapability = 0
5     h2250Capability = 0
6      maximumAudioDelayJitter = 60
6      receiveMultipointCapability = 0
7       multicastCapability = 0
7       multiUniCastConference = 0
7       mediaDistributionCapability = 0
8        * = 0
9         centralizedControl = 0
9         distributedControl = 0
9         centralizedAudio = 0
9         distributedAudio = 0
9         centralizedVideo = 0
9         distributedVideo = 0

6      transmitMultipointCapability = 0
7       multicastCapability = 0
7       multiUniCastConference = 0
7       mediaDistributionCapability = 0
8        * = 0
9         centralizedControl = 0
9         distributedControl = 0
9         centralizedAudio = 0
9         distributedAudio = 0
9         centralizedVideo = 0
9         distributedVideo = 0

6      receiveAndTransmitMultipointCapability = 0
7       multicastCapability = 0
7       multiUniCastConference = 0
7       mediaDistributionCapability = 0
8        * = 0
9         centralizedControl = 0
9         distributedControl = 0
9         centralizedAudio = 0
9         distributedAudio = 0
9         centralizedVideo = 0
9         distributedVideo = 0

6      mcCapability = 0
7       centralizedConferenceMC = 0
7       decentralizedConferenceMC = 0
6      rtcpVideoControlCapability = 0
6      mediaPacketizationCapability = 0
7       h261aVideoPacketization = 0

4    capabilityTable = 0
5     * = 0
6      capabilityTableEntryNumber = 7111
6      capability = 0
7       receiveAudioCapability = 0
8        g711Ulaw64k = 60

5     * = 0
6      capabilityTableEntryNumber = 7110
6      capability = 0
7       receiveAudioCapability = 0
8        g711Alaw64k = 60

5     * = 0
```

```
6       capabilityTableEntryNumber = 728
6       capability = 0
7        receiveAudioCapability = 0
8         g728 = 60

5      * = 0
6       capabilityTableEntryNumber = 261
6       capability = 0
7        receiveVideoCapability = 0
8         h261VideoCapability = 0
9          qcifMPI = 1
9          cifMPI = 1
9          temporalSpatialTradeOffCapability = 0
9          maxBitRate = 600
9          stillImageTransmission = 0

5      * = 0
6       capabilityTableEntryNumber = 263
6       capability = 0
7        receiveVideoCapability = 0
8         h263VideoCapability = 0
9          sqcifMPI = 1
9          qcifMPI = 1
9          cifMPI = 1
9          maxBitRate = 1000
9          unrestrictedVector = 0
9          arithmeticCoding = 0
9          advancedPrediction = 0
9          pbFrames = 0
9          temporalSpatialTradeOffCapability = 0
9          errorCompensation = 0


5      * = 0
6       capabilityTableEntryNumber = 7231
6       capability = 0
7        receiveAudioCapability = 0
8         g7231 = 0
9          maxAl-sduAudioFrames = 8
9          silenceSuppression = 0


5      * = 0 # Sequence
6       capabilityTableEntryNumber = 120 # INTEGER [1..65535]
6       capability = 0
7        receiveAndTransmitDataApplicationCapability = 0 # Sequence
8         application = 0
9          t120 = 0
10           separateLANStack = 0 # NULL [0..0]
8         maxBitRate = 1000 # INTEGER [0..4294967295]


4     capabilityDescriptors = 0
5      * = 0
6       capabilityDescriptorNumber = 0
6       simultaneousCapabilities = 0
7        * = 0
8         * = 7111
8         * = 7110
8         * = 7231
8         * = 728
7        * = 0
8         * = 261
8         * = 263
7        * = 0
8         * = 120
```

```
channels = 0
 * = 0
  name = 'g711Alaw64k'
  dataType = 0
   audioData = 0
    g711Alaw64k = 60

 * = 0
  name = 'g711Ulaw64k'
  dataType = 0
   audioData = 0
    g711Ulaw64k = 60

 * = 0
  name = 'g728'
  dataType = 0
   audioData = 0
    g728 = 60

 * = 0
  name = 'g7231'
  dataType = 0
   audioData = 0
    g7231 = 0
     maxAl-sduAudioFrames = 8
     silenceSuppression = 0

 * = 0
  name = 'h261VideoCapability'
  dataType = 0
   videoData = 0
    h261VideoCapability = 0
     qcifMPI = 1
     cifMPI = 1
     temporalSpatialTradeOffCapability = 0
     maxBitRate = 600
     stillImageTransmission = 0

 * = 0
  name = 'h263VideoCapability'
  dataType = 0
   videoData = 0
    h263VideoCapability = 0
     sqcifMPI = 1
     qcifMPI = 1
     cifMPI = 1
     maxBitRate = 1000
     unrestrictedVector = 0
     arithmeticCoding = 0
     advancedPrediction = 0
     pbFrames = 0
     temporalSpatialTradeOffCapability = 0
     errorCompensation = 0

 * = 0
  name = 't120'
  dataType = 0
   data = 0
    application = 0
     t120 = 0
      separateLANStack = 0
    maxBitRate = 1000

[MWIACTIVATE_SR]
```

```
servedUserNr=serveduser@example.com
basicService=0815
msgCentreId=4711
nbOfMessages=1
originatingNr=originating@example.com
timestamp=19970621194530
priority=7

[MWIDEACTIVATE_SR]
servedUserNr=serveduser@example.com
basicService=0815
msgCentreId=4711
callbackReq=1

[MWIINTERROGATE_SR]
servedUserNr=serveduser@example.com
basicService=0815
msgCentreId=4711
callbackReq=1

[MWIACK_SR]
basicService=0815
msgCentreId=4711
nbOfMessages=1
originatingNr=originating@example.com
timestamp=19970621194530
priority=7

[MWIREJ_SR]
Error=8

#include <rvcommon.h>
#include <msg.h>
#include <ms.h>
#include <pdlproc.h>
#include <stkutils.h>
#include <rtree.h>
#include <pdlraw.h>
#include <cmintr.h>
#include "h450api.h"
#include "h450struct.h"


static LOCAL_STRUCT_PTR control;
static cmElem* app;
static char string[2048];     //string, which carries the H.450
message
static unsigned int n;                    // counter to step through
the string


RVAPI void CALLCONV initAPI(MYHANDLE *ahandle)
{
    control = (LOCAL_STRUCT_PTR)calloc(1, sizeof(LOCAL_STRUCT));
    *ahandle = (MYHANDLE *)control;
}


RVAPI void CALLCONV endAPI()
{
      if (control)
      {
            free(control);
      }
}
```

```
RVAPI void CALLCONV setCallback(MYHANDLE ahandle, CALLBACK_T
callback)
{
LOCAL_STRUCT_PTR local = (LOCAL_STRUCT_PTR)ahandle;

    local->callback = callback;
}


RVAPI void CALLCONV initH450Root(char * configfile, MYHANDLE
*rootshandle)
{
      char pdlName[280] = "*RADVH450";
      pdlInitNumbers numbers =
    {
        /*vtNodeCount    */2700,
        /*channels       */100,
        /*chanDescs  5    */5,
        /*messages       */50,
        /*tpktChans      */5,
        /*udpChans       */3,
        /*protocols      */40,
        /*maxProcs       */40,
      /*maxBuffSize   */2048,
    };
      app=(cmElem*)calloc(1,sizeof(cmElem));


      msOpen();

      app->hCfg=ciConstruct(configfile);
      ciGetString(app-
>hCfg,"system.pdlName",pdlName,sizeof(pdlName));


      pdlInit((HPdlAProtocol)app,&(app-
>hsProtocol),(newProtocolEH)newH450RootChild,( char*)pdlName,app-
>hCfg,&numbers);

      *rootshandle=(MYHANDLE*)(app->hsProtocol);


      pdlStart(app->hsProtocol);
}


RVAPI void CALLCONV endH450Root()
{
      if (app)
    {
            pdlEnd(app->hsProtocol);
            ciDestruct(app->hCfg);
            msClose();
            free(app);
    }
}

RVAPI void CALLCONV createH450protocol(char * machinename, CALLBACK_T
rootshandle, CALLBACK_T ahandle, MYHANDLE *h450shandle)
// create a new protocol under the root
{
      cmElem* app=(cmElem*)rootshandle;
```

```
	// create new protocol: runs the create block of the specified
machine
	pdlCreateProtocol((HPdlSProtocol)app,(HPdlAProtocol)ahandle,
	(HPdlSProtocol*)h450shandle, machinename,
(newProtocolEH)newH450Child,(newMessageEH)newH450Message,FALSE);
}


RVAPI void CALLCONV closeH450protocol(CALLBACK_T mglinkershandle)
{
	// close the protocol for the messagelinker
	pdlCloseProtocol((HPdlSProtocol)mglinkershandle);
}

int newH450Message(HPdlAProtocol haProtocol, HPdlSProtocol
hsProtocol, int   message)
{
	HPST synH;
	HPVT valH;
	int stNodeId;           // the syntaxtree nodeId
	int fieldId;            // the fieldId of the node
	int vtNodeId;           // the valuetree nodeId
	char rootname[256]; // the rootname of the syntaxtree root node

	LOCAL_STRUCT_PTR local = (LOCAL_STRUCT_PTR)haProtocol;
	memset(string,0,sizeof(string));


	valH = pdlGetValTree((HPdlSProtocol)hsProtocol);


	vtNodeId=pvtGetNodeIdByPath(valH, message,"");

		synH=pvtGetSynTree(valH,vtNodeId);


	pvtGet(valH,vtNodeId,NULL,&stNodeId,NULL,NULL);


	pstGetRootName(synH, sizeof(rootname), rootname);


	fieldId=pstGetFieldId(synH, rootname);

	strcpy(string, "############################ H450-Message
############################");
	local->callback(string);
	strcpy(string, "");


	recursion(haProtocol, valH, synH, stNodeId, vtNodeId, fieldId,
0);

	// enable to send message in on esting: local-
>callback(string);

	return 0;

}


char recursion(HPdlAProtocol haProtocol, HPVT valH, HPST synH, int
stNode, int vtNode, INT32 fieldId, int tab)

{
	LOCAL_STRUCT_PTR local = (LOCAL_STRUCT_PTR)haProtocol;
```

```
        pstChild child;
        char fieldName[256];
        char fieldString[256];
        int stNodeIdChild;      // the syntaxtree nodeId of the Child
        int vtNodeIdChild;      // the valuetree nodeId of the Child
        int value;              // the value of the number stored in
the node or the leghth of the string stored in the node
        BOOL isString;          // TRUE if a string is stored in the
node
        int i;                          // counter

        vtNodeIdChild=vtNode;
        stNodeIdChild=stNode;

        // get the value stored in the node
        pvtGet(valH,vtNodeIdChild,NULL,NULL,&value,&isString);


        // insert the tabs
        for (i=0 ; i<tab; i++)
        {
              strcat(string, "    ");
        }

        // get the fieldname of the syntaxtree node
        pstGetFieldName(synH, fieldId, sizeof(fieldName), fieldName);

        strcat(string, "<");
        strcat(string, fieldName);
        strcat(string, ">");

        if (value<=0)
        {
              // Begin: remove to send message in one string
              local->callback(string);
              strcpy(string, "");
              // End: remove to send message in one string

        }

        // reads out the value, if a value is stored in the valuetree
node
        if (value>0)
                    {
                          // read out the string, if a string is stored
in the node
                          if (isString == TRUE)
                          {
                                pvtGetString(valH, vtNodeIdChild,
value, fieldString);
                                strcat(string, fieldString);
                          }

                          // read out the number, if a number is stored
in the node
                          else
                          {
                                // convert number to string
                                _itoa(value, fieldString, 10);
                                strcat(string, fieldString);
                          }
                          local->callback(string);
                          strcpy(string, "");
                    }

        // if there are any children of the current node , the function
```

```
	// calls itself with the stNode, vtNode and the fieldId of each child, and with
	// tab increased by one
	if (pstGetNumberOfChildren(synH, stNodeIdChild)>0)
		{

		for (i=1; i<pstGetNumberOfChildren(synH, stNodeIdChild)+1; i++)
			{
				pstGetChild(synH, stNodeIdChild, i, &child );

				// function only calls itself, when the child exists in the valuetree
				if (pvtGetChild(valH, vtNode, child.fieldId, &vtNodeIdChild)>0)
				{
					recursion(haProtocol, valH, synH, child.nodeId, vtNodeIdChild, child.fieldId, tab+1);
				}
			}

		}

	// if NodeType is Sequence of or Set of, the function also searches for
	// children of the Sequence of XXX / Set of XXX node.
	if (pstGetNodeType(synH, stNode)==19 || pstGetNodeType(synH, stNode)==20)
		{
		stNodeIdChild=pstGetNodeOfId(synH, stNode);
		vtNode=pvtChild(valH, vtNode);

		for (i=1; i<pstGetNumberOfChildren(synH, stNodeIdChild)+1; i++)
			{
				pstGetChild(synH, stNodeIdChild, i, &child );

				// function only calls itself, when the child exists in the valuetree
				if (pvtGetChild(valH, vtNode, child.fieldId, &vtNodeIdChild)>0)
				{
					recursion(haProtocol, valH, synH, child.nodeId, vtNodeIdChild, child.fieldId, tab+1);
				}
			}
		}

	// insert the tabs
	for (i=0 ; i<tab; i++)
	{
		strcat(string, "    ");
	}

	strcat(string, "</");
	strcat(string, fieldName);
	strcat(string, ">");

	// Begin: remove to send message in one string
	local->callback(string);
	strcpy(string, "");
	// End: remove to send message in one string

	return 0;
}
```

```
int newH450RootChild(
            IN      HPdlAProtocol           haProtocolParent,
            IN      HPdlSProtocol           hsProtocol,
            OUT     HPdlAProtocol*          lphaProtocol,
            IN      char*               protocolName,
            OUT     newProtocolEH*          newSessionP,
            OUT     newMessageEH*           newCallMsgP,
            OUT     BOOL*               selfDestructing)
{
    *newSessionP=(newProtocolEH)newH450RootChild;
    *newCallMsgP=newH450Message;
    *lphaProtocol=haProtocolParent;
    *selfDestructing=TRUE;
    return 0;
}


int newH450Child(
            IN      HPdlAProtocol           haProtocolParent,
            IN      HPdlSProtocol           hsProtocol,
            OUT     HPdlAProtocol*          lphaProtocol,
            IN      char*               protocolName,
            OUT     newProtocolEH*          newSessionP,
            OUT     newMessageEH*           newCallMsgP,
            OUT     BOOL*               selfDestructing)
{
    *newSessionP=(newProtocolEH)newH450Child;
    *newCallMsgP=newH450Message;
    *lphaProtocol=haProtocolParent;
    *selfDestructing=TRUE;
    return 0;
}


RVAPI void CALLCONV sendmessageH450(char * message, CALLBACK_T
h450shandle)
{
      int nodeId;
      cmElem*
app=(cmElem*)pdlGetAHandle(pdlRoot((HPdlSProtocol)h450shandle));

      // get the handle of the valuetree
      app->hVal = pdlGetValTree(app->hsProtocol);

      // create a new value tree based on "h4507app2pdl"
      nodeId=pvtAddRoot(app->hVal,pdlGetSynTreeByRootName(app-
>hsProtocol,(char*)"h4507app2pdl"),0,NULL);
      n=0; /* set counter to 0*/

      // call function to fill in the fields of the valuetree sent in
message
      fillin(message, "", app->hVal, nodeId);
      pdlSendMessage((HPdlSProtocol)h450shandle, nodeId);
}


char fillin (char * message, char * rootpath, HPVT hVal, int nodeId)
{
int length;
int nodeIdTemp;
char path[2048];
char fieldvalue[256];
memset(path,0,sizeof(path));
memset(fieldvalue,0,sizeof(fieldvalue));
```

```
strcpy(path, rootpath);
length = strlen(message);

while (message[n] == '<' && message[n+1] != '/')
{
      n++;
      fillin(message, path, hVal, nodeId);
      if (message[n] == 0 || (message[n] == '<' && message[n+1] ==
'/')) return 0;
}


      if (message[n] != '/')
      {
            n--;
            if (n!=0) strcat(path, ".");
            n++;
            while (message[n] != '>')
            {
                  strncat(path, message+n, 1);
                  n++;
            }
            n++;

            if (message[n] != '<')
            {
                  strcpy(fieldvalue, "");
                  while (message[n] != '<')
                  {
                        strncat(fieldvalue, message+n, 1);
                        n++;
                  }
                  nodeIdTemp =
pvtBuildByPath(hVal,nodeId,path,sizeof(fieldvalue),fieldvalue);
                  n++;
            }

            else
            {
                  nodeIdTemp =
pvtBuildByPath(hVal,nodeId,path,0,NULL);
                  if (message[n] == '<' && message[n+1] != '/')
                  {
                        fillin(message, path, hVal, nodeId);
                        n++;
                  }
                  else n++;
            }
      }

      if (message[n] == '/')
      {
            n--;
            while (message[n] != '>')
            {
                  n++;
            }
            n++;
            return 0;
      }
}
#ifndef H450API_H
#define H450API_H

#include <cm.h>
```

```
/* Typedefs */

typedef void *MYHANDLE;

typedef void (*CALLBACK_T)(char*);

typedef struct
{
    CALLBACK_T callback;
} LOCAL_STRUCT, *LOCAL_STRUCT_PTR;

/* Function prototypes */

RVAPI void CALLCONV initAPI(MYHANDLE *ahandle);
RVAPI void CALLCONV endAPI();
RVAPI void CALLCONV setCallback(MYHANDLE ahandle, CALLBACK_T
callback);
RVAPI void CALLCONV initH450Root(char * configfile, MYHANDLE
*rootshandle);
RVAPI void CALLCONV endH450Root();
RVAPI void CALLCONV sendmessageH450(char * string, CALLBACK_T
h450shandle);
RVAPI void CALLCONV initializeH(void);
RVAPI void CALLCONV createH450protocol(char * machinename, CALLBACK_T
rootshandle, CALLBACK_T ahandle, MYHANDLE *h450shandle);
RVAPI void CALLCONV closeH450protocol(CALLBACK_T mglinkershandle);

#endif /* H450API_H */
#ifndef H450_H
#define H450_H

#include <cm.h>


/* Typedefs */

typedef struct
{
    HRTREE      hrTree;  /* the tree of protocols*/
    HDRV            hTpktChan;/* ther are 2 types of external
channels */
    HDRV            hUdpChan;
    HPDLCHANDESC    hApplChan;/* application channel internal
channel,implemented by pdlapi*/
    HPDL            hPdl;     /* pointer to global structure in
pdlproc layer*/
    HCHN            hChn;    /* handle passed to all channels
functions*/
    HVALT           hVal;
    newCallbackEH   newCallback; /* global callback,that calls before
all callbacks
        created for X */
    setAppBufferEH  setAppBuffer;/* notify the appl about new message
on the one of the
        application channels*/
    HCFG            hCfg; /* configuration handle */
    HPDLRAW         hRaw; /* handle to pdl file */
    int               msa; /* pdlapi printing */
} pdlApp; /* one for all protocols contains all global values*/

typedef struct
{
    pdlApp*       app;
    int                 id; /*id in the protocol tree  */
    HPdlAProtocol haProtocol; /* application handle*/
    HPDLP         hProc; /* pointer to machine */
```

```
    UINT32              key;  /*protocol parameter is stored as tree
*/
    newProtocolEH       newProtocol;/* called when new child protocol
is created*/
    newMessageEH  newMessage; /* protocol receives messages from
machine via this cllback */
    BOOL                passive; /* set if tried to close protocol
and not succeded because of childs*/
    BOOL                selfDestructing;/* can parent close child
protocol without closing this child from
                                application*/
} pdlElem; /*protocol structure */

/* Function prototypes */

int newH450RootChild(
          IN      HPdlAProtocol           haProtocolParent,
          IN      HPdlSProtocol           hsProtocol,
          OUT     HPdlAProtocol*          lphaProtocol,
          IN      char*              protocolName,
          OUT     newProtocolEH*          newSessionP,
          OUT     newMessageEH*           newCallMsgP,
          OUT     BOOL*              selfDestructing


          );

int newH450Child(
          IN      HPdlAProtocol           haProtocolParent,
          IN      HPdlSProtocol           hsProtocol,
          OUT     HPdlAProtocol*          lphaProtocol,
          IN      char*              protocolName,
          OUT     newProtocolEH*          newSessionP,
          OUT     newMessageEH*           newCallMsgP,
          OUT     BOOL*              selfDestructing


          );

int newH450Message(
             IN   HPdlAProtocol           haProtocol,
             IN   HPdlSProtocol           hsProtocol,
             IN   int                message
             );
char recursion(HPdlAProtocol haProtocol, HPVT valH, HPST synH, int
stNode, int vtNode, INT32 fieldId, int tab);
char fillin (char * dummy, char * dummypath, HPVT hVal, int nodeId);

#endif /* H450_H *//


#include <rvcommon.h>
#include <msg.h>
#include <ms.h>
#include <pdlproc.h>
#include <stkutils.h>
#include <rtree.h>
#include <pdlraw.h>
#include <cmintr.h>
#include <emanag.h>
#include "h450api.h"
#include "h450struct.h"

#include <ssintr.h>


static LOCAL_STRUCT_PTR control;
```

```
static ssElem* app;
static char string[2048];
static char buffer[2048];
static unsigned int n;

RVAPI void CALLCONV tpInitAPI(MYHANDLE *ahandle)
{
    control = (LOCAL_STRUCT_PTR)calloc(1, sizeof(LOCAL_STRUCT));
    *ahandle = (MYHANDLE *)control;
}


RVAPI void CALLCONV tpEndAPI()
{
      if (control)
      {
            free(control);
      }
}

RVAPI void CALLCONV tpSetCallback(MYHANDLE ahandle, CALLBACK_T
callback)
{
LOCAL_STRUCT_PTR local = (LOCAL_STRUCT_PTR)ahandle;

    local->callback = callback;
}

RVAPI int tpNewH450Message(HPdlAProtocol haProtocol, HPdlSProtocol
hsProtocol, int   message)
{
      HPST synH;
      HPVT valH;
            int stNodeId;
      int fieldId;
      int vtNodeId;                  char rootname[256];
      LOCAL_STRUCT_PTR local = (LOCAL_STRUCT_PTR)haProtocol;
      memset(string,0,sizeof(string));
      memset(buffer,0,sizeof(buffer));

            valH = pdlGetValTree((HPdlSProtocol)hsProtocol);


      //vtNodeId=pvtGetNodeIdByPath(valH,
message,"h4501SupplemetaryService"); /* insert for h.450.1 */
      vtNodeId=pvtGetNodeIdByPath(valH, message,"");/* remove for
h.450.1 */

            synH=pvtGetSynTree(valH,vtNodeId);

            pvtGet(valH,vtNodeId,NULL,&stNodeId,NULL,NULL);

            pstGetRootName(synH, sizeof(rootname), rootname);


      fieldId=pstGetFieldId(synH, rootname);

      strcpy(string, "#############  received H450-Message
#############");/* remove for h.450.1 */
      local->callback(string);/* remove for h.450.1 */
      strcpy(string, "");

      //emEncode((HVALT)valH,vtNodeId,(BYTE*)buffer,sizeof(buffer),&e
ncoded); /* insert for h.450.1 */
      //local->callback(buffer); /* insert for h.450.1 */
```

```
      tpReadOutVT(haProtocol, valH, synH, stNodeId, vtNodeId,
fieldId, 0); /* remove for h.450.1 */

      // enable to send message in one string: local-
>callback(string);

      return 0;
}

RVAPI int CALLCONV tpCreateService(
                              IN      HSSAPP      hSSApp,
                              OUT     HSSSERVICE* hSSServ,
                              IN      HSSAPPSERVICE hSSaServ,
                        IN          char*   serviceName)
{
    int ret;
    ssElem* app=(ssElem*)hSSApp;

    ret = pdlCreateProtocol(app->hsProtocol,(HPdlAProtocol)hSSaServ,
      (HPdlSProtocol*)hSSServ,serviceName,
NULL,tpNewH450Message,FALSE);
    return ret;
}


char tpReadOutVT(HPdlAProtocol haProtocol, HPVT valH, HPST synH, int
stNode, int vtNode, INT32 fieldId, int tab)

{
      LOCAL_STRUCT_PTR local = (LOCAL_STRUCT_PTR)haProtocol;
      pstChild child;
      char fieldName[256];
      char fieldString[256];
      int stNodeIdChild;
      int vtNodeIdChild;
      int value;
      BOOL isString;
      int i;

      vtNodeIdChild=vtNode;
      stNodeIdChild=stNode;


      pvtGet(valH,vtNodeIdChild,NULL,NULL,&value,&isString);


            for (i=0 ; i<tab; i++)
      {
            strcat(string, "    ");
      }

            pstGetFieldName(synH, fieldId, sizeof(fieldName),
fieldName);

      strcat(string, "<");
      strcat(string, fieldName);
      strcat(string, ">");


      if (value>0 && pstGetNodeType(synH, stNode)<15) //value>=0 ???
      {

            if (isString == TRUE)
```

```
                {
                        pvtGetString(valH, vtNodeIdChild, value,
fieldString);
                        strncat(string, fieldString, value);
            strcpy(fieldString, "");
                }


                else
                {

                        _itoa(value, fieldString, 10);
                        strcat(string, fieldString);
            strcpy(fieldString, "");
                }
                local->callback(string);
                strcpy(string, "");
        }
      else
        {

                local->callback(string);
                strcpy(string, "");

        }



        if (pstGetNumberOfChildren(synH, stNodeIdChild)>0)
                {

                for (i=1; i<pstGetNumberOfChildren(synH,
stNodeIdChild)+1; i++)
                        {
                              pstGetChild(synH, stNodeIdChild, i, &child );

                                                    if (pvtGetChild(valH,
vtNode, child.fieldId, &vtNodeIdChild)>0)
                                {
                                        tpReadOutVT(haProtocol, valH, synH,
child.nodeId, vtNodeIdChild, child.fieldId, tab+1);
                                }
                        }

                }


        if (pstGetNodeType(synH, stNode)==19 || pstGetNodeType(synH,
stNode)==20)
                {
                stNodeIdChild=pstGetNodeOfId(synH, stNode);
                vtNode=pvtChild(valH, vtNode);

                for (i=1; i<pstGetNumberOfChildren(synH,
stNodeIdChild)+1; i++)
                        {
                              pstGetChild(synH, stNodeIdChild, i, &child );

                                                    if (pvtGetChild(valH,
vtNode, child.fieldId, &vtNodeIdChild)>0)
                                {
                                        tpReadOutVT(haProtocol, valH, synH,
child.nodeId, vtNodeIdChild, child.fieldId, tab+1);
                                }
                        }
                }
```

```
            for (i=0 ; i<tab; i++)
        {
            strcat(string, "    ");
        }

        strcat(string, "</");
        strcat(string, fieldName);
        strcat(string, ">");

            local->callback(string);
        strcpy(string, "");

        return 0;
}

RVAPI void CALLCONV tpSendMessageH4507(char * message, HSSSERVICE
h450shandle)

{
        int nodeId;
        ssElem*
app=(ssElem*)pdlGetAHandle(pdlRoot((HPdlSProtocol)h450shandle));

            app->hVal = pdlGetValTree(app->hsProtocol);

            nodeId=pvtAddRoot(app->hVal,pdlGetSynTreeByRootName(app-
>hsProtocol,(char*)"h4507app2pdl"),0,NULL); /* remove for h.450.1 */

        n=0; /* set counter to 0*/


        tpPopulateVT(message, "", app->hVal, nodeId);
        pdlSendMessage((HPdlSProtocol)h450shandle, nodeId);
}

RVAPI void CALLCONV tpSendMessageH4501(char * message, HSSSERVICE
h450shandle)

{
        int nodeId;
        ssElem*
app=(ssElem*)pdlGetAHandle(pdlRoot((HPdlSProtocol)h450shandle));

            app->hVal = pdlGetValTree(app->hsProtocol);


        nodeId=pvtAddRoot(app->hVal,pdlGetSynTreeByRootName(app-
>hsProtocol,(char*)"h4501AppPrimitive"),0,NULL); /* insert for
h.450.1 */

        n=0; /* set counter to 0*/


        tpPopulateVT(message, "", app->hVal, nodeId);
        pdlSendMessage((HPdlSProtocol)h450shandle, nodeId);
}

RVAPI void CALLCONV tpSendMessageH450x(char * message, HSSSERVICE
h450shandle)

{
        int nodeId;
        ssElem*
app=(ssElem*)pdlGetAHandle(pdlRoot((HPdlSProtocol)h450shandle));

            app->hVal = pdlGetValTree(app->hsProtocol);
```

```
            nodeId=pvtAddRoot(app->hVal,pdlGetSynTreeByRootName(app-
>hsProtocol,(char*)"h450xAppPrimitive"),0,NULL); /* insert for
h.450.1 */

      n=0; /* set counter to 0*/


      tpPopulateVT(message, "", app->hVal, nodeId);
      pdlSendMessage((HPdlSProtocol)h450shandle, nodeId);
}

int chrn2bmp1(char *str, int maxStrLen, BYTE* bmpStr)
{
    int i, i2;

    for (i = 0, i2 = 0; i < maxStrLen; i++, i2 += 2)
    {
      bmpStr[i2 + 0] = 0;
      bmpStr[i2 + 1] = str[i];
    }

    return i2;
}

int chr2bmp1(char *str, BYTE* bmpStr)
{
    return chrn2bmp1(str, strlen(str), bmpStr);
}


char tpPopulateVT (char * message, char * rootpath, HPVT hVal, int
nodeId)
{
UINT32 fieldvalueOctet;
int length;
int fieldvaluesize;
int fieldvalueInt;
int fieldvaluelength;
int bmpLength;
int nodeIdTemp;
char bmpStr[512];
char path[2048];
char fieldvalue[256];
char stringOrInt;
memset(path,0,sizeof(path));
memset(fieldvalue,0,sizeof(fieldvalue));
strcpy(path, rootpath);
length = strlen(message);

while (message[n] == '<' && message[n+1] != '/')
{
      n++;
      tpPopulateVT(message, path, hVal, nodeId);
      if (message[n] == 0 || (message[n] == '<' && message[n+1] ==
'/')) return 0;
}


      if (message[n] != '/')
      {
            n--;
            if (n!=0) strcat(path, ".");
            n++;
            while (message[n] != '>')
            {
                  strncat(path, message+n, 1);
                  n++;
```

```
                }
                n++;

                if (message[n] != '<')
                {
                        stringOrInt=message[n];
                        n++;
                        fieldvaluesize=0;
                        strcpy(fieldvalue, "");
                        while (message[n] != '<')
                        {
                                strncat(fieldvalue, message+n, 1);
                                n++;
                                fieldvaluesize++;
                        }
                        fieldvalue[fieldvaluesize]='\0';
                        fieldvaluesize++;
                        fieldvaluelength=strlen(fieldvalue);
                        if (stringOrInt=='s')
                        {
                                nodeIdTemp =
pvtBuildByPath(hVal,nodeId,path,fieldvaluelength,fieldvalue);
                        }
                        if (stringOrInt=='b')
                        {
                                bmpLength=chr2bmp1(fieldvalue,
(BYTE*)bmpStr);
                                nodeIdTemp =
pvtBuildByPath(hVal,nodeId,path,bmpLength,bmpStr);
                        }
                        if (stringOrInt=='i')
                        {
                                fieldvalueInt = atoi(fieldvalue);
                                nodeIdTemp =
pvtBuildByPath(hVal,nodeId,path,fieldvalueInt,NULL);
                        }
                        if (stringOrInt=='o')
                        {
                                fieldvalueOctet = atoi(fieldvalue);
                                nodeIdTemp =
pvtBuildByPath(hVal,nodeId,path,sizeof(UINT32),(fieldvalueOctet)?((ch
ar*)&fieldvalueOctet):(char*)"\0\0\0");
                        }
                        n++;
                }

                else
                {
                        nodeIdTemp =
pvtBuildByPath(hVal,nodeId,path,0,NULL);
                        if (message[n] == '<' && message[n+1] != '/')
                        {
                                tpPopulateVT(message, path, hVal, nodeId);
                                n++;
                        }
                        else n++;
                }
        }

        if (message[n] == '/')
        {
                n--;
                while (message[n] != '>')
                {
                        n++;
                }
                n++;
```

```
            return 0;
        }
}

#ifndef H450API_H
#define H450API_H

#include <cm.h>
#include <pdlapi.h>
#include <h450.h>


/* Typedefs */

typedef void *MYHANDLE;

typedef void (*CALLBACK_T)(char*);

typedef struct
{
    CALLBACK_T callback;
} LOCAL_STRUCT, *LOCAL_STRUCT_PTR;


RVAPI void CALLCONV tpInitAPI(MYHANDLE *ahandle);
RVAPI void CALLCONV tpEndAPI();
RVAPI void CALLCONV tpSetCallback(MYHANDLE ahandle, CALLBACK_T
callback);
RVAPI void CALLCONV tpSendMessageH4501(char * string, HSSSERVICE
h450shandle);
RVAPI void CALLCONV tpSendMessageH450x(char * string, HSSSERVICE
h450shandle);
RVAPI void CALLCONV tpSendMessageH4507(char * string, HSSSERVICE
h450shandle);
RVAPI int  tpNewH450Message(HPdlAProtocol haProtocol, HPdlSProtocol
hsProtocol, int   message);
RVAPI int  CALLCONV tpCreateService(HSSAPP hSSApp, HSSSERVICE*
hSSServ, HSSAPPSERVICE hSSaServ, char*    serviceName);

#endif /* H450API_H */
#include <windows.h>
static HINSTANCE hInst;
HINSTANCE getHInst(void)
{
      return hInst;
}

#ifdef WIN32

BOOL WINAPI DllMain(
    IN HINSTANCE hinstDLL,     // handle to DLL module
    IN DWORD fdwReason,        // reason for calling function
    IN LPVOID lpvReserved      // reserved
   )
{
      switch(fdwReason)
      {
            case DLL_PROCESS_ATTACH:
                  hInst=hinstDLL;
                  break;
            case DLL_PROCESS_DETACH:
                  break;
      }
      return TRUE;
}
```

```
#else
int FAR PASCAL LibMain(HANDLE hInstance, WORD wDataSegment,
                       WORD wHeapSize, LPSTR lpszCmdLine)
{
      if (hInst != NULL)
            return FALSE;

      hInst = hInstance;
      return TRUE;
}

int FAR PASCAL __export WEP (int bSystemExit)
{
      hInst = NULL;
      return 0;
}

#endif
#ifndef H450_H
#define H450_H

#include <cm.h>


char tpReadOutVT(HPdlAProtocol haProtocol, HPVT valH, HPST synH, int
stNode, int vtNode, INT32 fieldId, int tab);
char tpPopulateVT (char * dummy, char * dummypath, HPVT hVal, int
nodeId);

#endif /* H450_H *//
```